UNIT-III ARRAYS AND STRINGS

ARRAYS

Introduction:

So far we have used only single variable name for storing one data item. If we need to store multiple copies of the same data then it is very difficult for the user. To overcome the difficulty a new data structure is used called arrays.

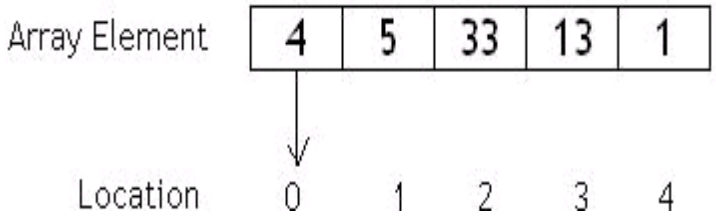
- An array is a linear and homogeneous data structure
- An array permits homogeneous data. It means that similar types of elements are stored contiguously in the memory under one variable name.
- An array can be declared of any standard or custom data type.

Example of an Array:

Suppose we have to store the roll numbers of the 100 students the we have to declare 100 variables named as roll1, roll2, roll3, ……. roll100 which is very difficult job. Concept of C programming arrays is introduced in C which gives the capability to store the 100 roll numbers in the contiguous memory which has 100 blocks and which can be accessed by single variable name.

1. C Programming Arrays is the **Collection of Elements**
2. C Programming Arrays is collection of the Elements of the **same data type.**
3. All Elements are stored in the **Contiguous memory**
4. All elements in the array are accessed using the subscript variable (index).

Pictorial representation of C Programming Arrays



The above array is declared as int a [5];

a[0] = 4; a[1] = 5; a[2] = 33; a[3] = 13; a[4] = 1;

In the above figure 4, 5, 33, 13, 1 are actual data items. 0, 1, 2, 3, 4 are index variables.

Index or Subscript Variable:

1. Individual data items can be accessed by the name of the array and an integer enclosed in square bracket called subscript variable / index

2. Subscript Variables helps us to identify the item number to be accessed in the contiguous memory.

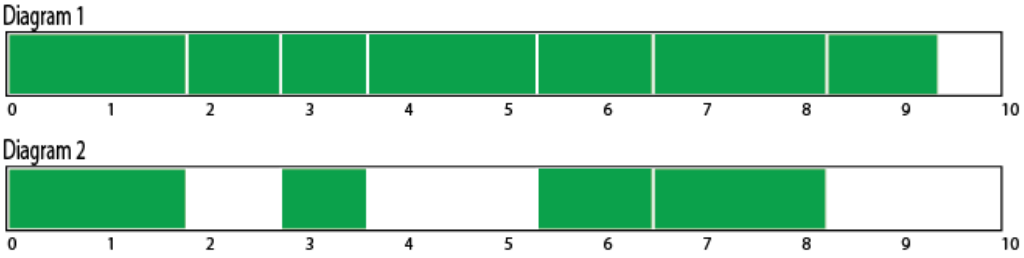
What is Contiguous Memory?

1. When Big Block of memory is reserved or allocated then that memory block is called as Contiguous Memory Block.
2. Alternate meaning of Contiguous Memory is continuous memory.
3. Suppose inside memory we have reserved 1000-1200 memory addresses for special purposes then we can say that these 200 blocks are going to reserve contiguous memory.

Contiguous Memory Allocation

1. Two registers are used while implementing the contiguous memory scheme. These registers are base register and limit register.
2. When OS is executing a process inside the main memory then content of each register are as

Register	Content of register
Base register	Starting address of the memory location where process execution is happening
Limit register	Total amount of memory in bytes consumed by process



Here diagram 1 represents the contiguous allocation of memory and diagram 2 represents non-contiguous allocation of memory.

3. When process try to refer a part of the memory then it will firstly refer the base address from base register and then it will refer relative address of memory location with respect to base address.

How to allocate contiguous memory?

1. Using static array declaration.
2. Using alloc ( ) / malloc ( ) function to allocate big chunk of memory dynamically.

Array Terminologies:

Size: Number of elements or capacity to store elements in an array. It is always mentioned in square brackets [ ].

Type: Refers to data type. It decides which type of element is stored in the array. It is also instructing the compiler to reserve memory according to the data type.

Base: The address of the first element is a base address. The array name itself stores address of the first element.

Index: The array name is used to refer to the array element. For example num[x], num is array and x is index. The value of x begins from 0.The index value is always an integer value.

Range: Value of index of an array varies from lower bound to upper bound. For example in num[100] the range of index is 0 to 99.

Word: It indicates the space required for an element. In each memory location, computer can store a data piece. The space occupation varies from machine to machine. If the size of element is more than word (one byte) then it occupies two successive memory locations. The variables of data type int, float, long need more than one byte in memory.

Characteristics of an array:

1. The declaration int a [5] is nothing but creation of five variables of integer types in memory instead of declaring five variables for five values.
2. All the elements of an array share the same name and they are distinguished from one another with the help of the element number.
3. The element number in an array plays a major role for calling each element.
4. Any particular element of an array can be modified separately without disturbing the other elements.
5. Any element of an array a[ ] can be assigned or equated to another ordinary variable or array variable of its type.
6. Array elements are stored in contiguous memory locations.

Array Declaration:

Array has to be declared before using it in C Program. Array is nothing but the collection of elements of similar data types.

Syntax: <data type> array name [size1][size2].....[sizen];

Syntax Parameter	Significance
Data type	Data Type of Each Element of the array
Array name	Valid variable name

Size	Dimensions of the Array

Array declaration requirements

Requirement	Explanation
Data Type	Data Type specifies the type of the array. We can compute the size required for storing the single cell of array.
Valid Identifier	Valid identifier is any valid variable or name given to the array. Using this identifier name array can be accessed.
Size of Array	It is maximum size that array can have.

What does Array Declaration tell to Compiler?

1. Type of the Array
2. Name of the Array
3. Number of Dimension
4. Number of Elements in Each Dimension

Types of Array

1. **Single Dimensional Array / One Dimensional Array**
2. **Multi Dimensional Array**

Single / One Dimensional Array:

1. Single or One Dimensional array is used to represent and store data in a linear form.
2. Array having only one subscript variable is called **One-Dimensional array**
3. It is also called as **Single Dimensional Array** or **Linear Array**

Single Dimensional Array Declaration and initialization:

Syntax for declaration: <data type> <array name> [size];

Examples for declaration: int iarr[3]; **char** carr[20]; **float** farr[3];

Syntax for initialization: <data type> <array name> [size] = {val1, val2, …, valn};

Examples for initialization:

int iarr[3] = {2, 3, 4};

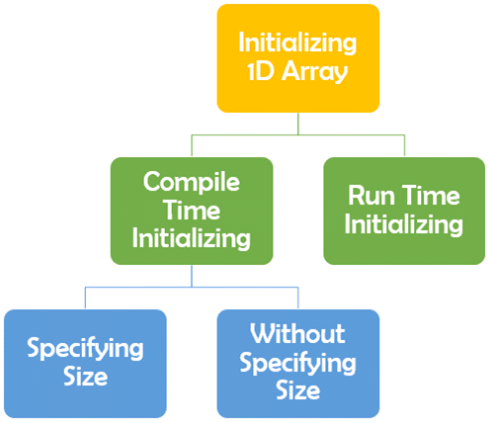
char carr[20] **=** “program”;

float farr[3] = {12.5, 13.5, 14.5};

Different Methods of Initializing 1-D Array

Whenever we declare an array, we initialize that array directly at compile time.

Initializing 1-D Array is called as compiler time initialization if and only if we assign certain set of values to array element before executing program. i.e. at compilation time.



Here we are learning the different ways of compile time initialization of an array.

Ways of Array Initializing 1-D Array:

1. Size is Specified Directly
2. Size is Specified Indirectly

Method 1: Array Size Specified Directly

In this method, we try to specify the Array Size directly.

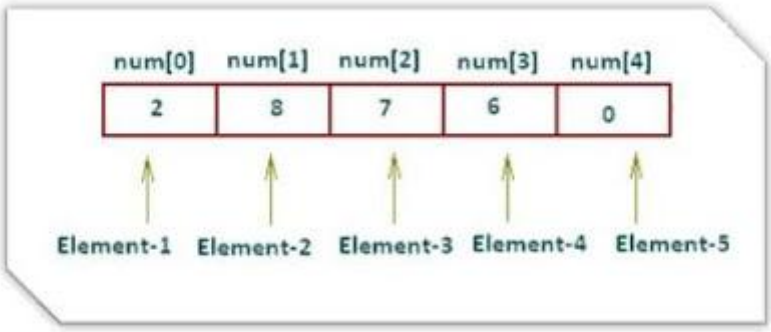
int num [5] = {2,8,7,6,0};

In the above example we have specified the size of array as 5 directly in the initialization statement. Compiler will assign the set of values to particular element of the array.

num[0] = 2; num[1] = 8; num[2] = 7; num[3] = 6; num[4] = 0;

As at the time of compilation all the elements are at specified position So This initialization scheme is Called as "**Compile Time Initialization**".

Graphical Representation:



Method 2: Size Specified Indirectly

In this scheme of compile time Initialization, We do not provide size to an array but instead we provide set of values to the array.

int num[ ] = {2,8,7,6,0};

Explanation:

1. Compiler Counts the Number Of Elements Written Inside Pair of Braces and Determines the Size of An Array.
2. After counting the number of elements inside the braces, The size of array is considered as 5 during complete execution.
3. This type of Initialization Scheme is also Called as "**Compile Time Initialization**"

Example Program

```
#include <stdio.h>
int main()
int num[] = {2,8,7,6,0};
int i;
for (i=0;i<5;i++) {
printf("\n Array Element num [%d] = %d",i, num[i]); }
return 0; }
```

Output:

```
Array Element num[0] = 2
Array Element num[1] = 8
Array Element num[2] = 7
Array Element num[3] = 6
Array Element num[4] = 0
```

Accessing Array

1. We all know that array elements are randomly accessed using the subscript variable.
2. Array can be accessed using array-name and subscript variable written inside pair of square brackets [ ].

Consider the below example of an array

In this example we will be accessing array like this

arr[3] = Forth Element of Array

arr[5] = Sixth Element of Array

whereas elements are assigned to an array using below way

arr[0] = 51; arr[1] = 32; arr[2] = 43; arr[3] = 24; arr[4] = 5; arr[5] =26;

Example Program1: Accessing array

```
#include<stdio.h>
#include<conio.h>
void main()
{
int arr[] = {51,32,43,24,5,26};
int i;
for(i=0; i<=5; i++) {
printf("\nElement at arr[%d] is %d",i,arr[i]);
}
getch();
}
```

```
Output:
Element at arr[0] is 51
Element at arr[1] is 32
Element at arr[2] is 43
Element at arr[3] is 24
Element at arr[4] is 5
Element at arr[5] is 26
```

How a[i] Works?

We have following array which is declared like int arr[] = { 51,32,43,24,5,26};

As we have elements in an array, so we have track of base address of an array. Below things are important to access an array.

Expression	Description	Example
arr	It returns the base address of an array	Consider 2000
*arr	It gives zeroth element of an array	51

Expression	Description	Example
*(arr+0)	It also gives zeroth element of an array	51
*(arr+1)	It gives first element of an array	32

So whenever we tried accessing array using arr[i] then it returns an element at the location*(arr + i)

Accessing array a[i] means retrieving element from address (a + i).

Example Program2: Accessing array

```
#include<stdio.h>
#include<conio.h>
void main()
{
int arr[] = {51,32,43,24,5,26};
int i;
for(i=0; i<=5; i++) {
printf("\n%d %d %d %d",arr[i],*(i+arr),*(arr+i),i[arr]);
}
getch();
}
```

Output:

```
51 51 51 51
32 32 32 32
43 43 43 43
24 24 24 24
5 5 5 5
26 26 26 26
```

Operations with One Dimensional Array

1. Deletion – Involves deleting specified elements form an array.
2. Insertion – Used to insert an element at a specified position in an array.
3. Searching – An array element can be searched. The process of seeking specific elements in an array is called searching.

4. Merging – The elements of two arrays are merged into a single one.
5. Sorting – Arranging elements in a specific order either in ascending or in descending order.

Example Programs:

1. C Program for deletion of an element from the specified location from an Array

```
#include<stdio.h>
int main() {
int arr[30], num, i, loc;
printf("\nEnter no of elements:");
scanf("%d", &num);
//Read elements in an array
printf("\nEnter %d elements :", num);
for (i = 0; i < num; i++) {
scanf("%d", &arr[i]);    }
//Read the location
printf("\nLocation of the element to be deleted :");
scanf("%d", &loc);
/* loop for the deletion  */
while (loc < num) {
arr[loc - 1] = arr[loc];
loc++;    }
num--;  // No of elements reduced by 1
//Print Array
for (i = 0; i < num; i++)
printf("\n %d", arr[i]);
return (0);
}
```

Output:

```
Enter no of elements: 5
Enter 5 elements: 3 4 1 7 8
```

```
Location of the element to be deleted: 3
3 4 7 8
```

2. C Program to delete duplicate elements from an array

```
int main() {
int arr[20], i, j, k, size;
printf("\nEnter array size: ");
scanf("%d", &size);
printf("\nAccept Numbers: ");
for (i = 0; i < size; i++)
scanf("%d", &arr[i]);
printf("\nArray with Unique list: ");
for (i = 0; i < size; i++) {
for (j = i + 1; j < size;) {
if (arr[j] == arr[i]) {
for (k = j; k < size; k++) {
arr[k] = arr[k + 1]; }
size--; }
else
j++; }
}
for (i = 0; i < size; i++) {
printf("%d ", arr[i]); }
return (0);
}
```

Output:

```
Enter array size: 5
Accept Numbers: 1 3 4 5 3
Array with Unique list: 1 3 4 5
```

3. C Program to insert an element in an array

```
#include<stdio.h>
int main() {
```

```
int arr[30], element, num, i, location;
printf("\nEnter no of elements:");
scanf("%d", &num);
for (i = 0; i < num; i++) {
scanf("%d", &arr[i]); }
printf("\nEnter the element to be inserted:");
scanf("%d", &element);
printf("\nEnter the location");
scanf("%d", &location);
//Create space at the specified location
for (i = num; i >= location; i--) {
arr[i] = arr[i - 1];   }
num++;
arr[location - 1] = element;
//Print out the result of insertion
for (i = 0; i < num; i++)
printf("n %d", arr[i]);
return (0);
}
Output:
Enter no of elements: 5
1 2 3 4 5
Enter the element to be inserted: 6
Enter the location: 2
1 6 2 3 4 5
```

4. C Program to search an element in an array

```
#include<stdio.h>
int main() {
int a[30], ele, num, i;
printf("\nEnter no of elements:");
scanf("%d", &num);
```

```
printf("\nEnter the values :");
for (i = 0; i < num; i++) {
scanf("%d", &a[i]);    }
//Read the element to be searched
printf("\nEnter the elements to be searched :");
scanf("%d", &ele);
//Search starts from the zeroth location
i = 0;
while (i < num && ele != a[i]) {
i++;   }
//If i < num then Match found
if (i < num) {
printf("Number found at the location = %d", i + 1);
}
else {
printf("Number not found"); }
return (0);
}
```

Output:

```
Enter no of elements: 5
11 22 33 44 55
Enter the elements to be searched: 44
Number found at the location = 4
```

5. C Program to copy all elements of an array into another array

```
#include<stdio.h>
int main() {
int arr1[30], arr2[30], i, num;
printf("\nEnter no of elements:");
scanf("%d", &num);
//Accepting values into Array
printf("\nEnter the values:");
```

```
for (i = 0; i < num; i++) {
scanf("%d", &arr1[i]);    }
/* Copying data from array 'a' to array 'b */
for (i = 0; i < num; i++) {
arr2[i] = arr1[i];   }
//Printing of all elements of array
printf("The copied array is:");
for (i = 0; i < num; i++)
printf("\narr2[%d] = %d", i, arr2[i]);
return (0);
}
```

Output:

```
Enter no of elements: 5
Enter the values: 11 22 33 44 55
The copied array is: 11 22 33 44 55
```

6. C program to merge two arrays in C Programming

```
#include<stdio.h>
int main() {
int arr1[30], arr2[30], res[60];
int i, j, k, n1, n2;
printf("\nEnter no of elements in 1st array:");
scanf("%d", &n1);
for (i = 0; i < n1; i++) {
scanf("%d", &arr1[i]); }
printf("\nEnter no of elements in 2nd array:");
scanf("%d", &n2);
for (i = 0; i < n2; i++) {
scanf("%d", &arr2[i]); }
i = 0;
j = 0;
k = 0;
```

```
// Merging starts
while (i < n1 && j < n2) {
if (arr1[i] <= arr2[j]) {
res[k] = arr1[i];
i++;
k++; }
else {
res[k] = arr2[j];
k++;
j++; }
}
/*Some elements in array 'arr1' are still remaining where as the array
'arr2' is exhausted*/
while (i < n1) {
res[k] = arr1[i];
i++;
k++; }
/*Some elements in array 'arr2' are still remaining where as the array
'arr1' is exhausted */
while (j < n2) {
res[k] = arr2[j];
k++;
j++; }
//Displaying elements of array 'res'
printf("\nMerged array is:");
for (i = 0; i < n1 + n2; i++)
printf("%d ", res[i]);
return (0);
}
Enter no of elements in 1st array: 4
11 22 33 44
```

```
Enter no of elements in 2nd array: 3
10 40 80
Merged array is: 10 11 22 33 40 44 80
```

Programs for Practice

1 C Program to display array elements with addresses
2 C Program for Reading and printing Array Elements
3 C Program to calculate Addition of All Elements in Array
4 C Program to find Smallest Element in Array
5 C Program to find Largest Element in Array
6 C Program to reversing an Array Elements

1. C Program to display array elements with addresses

```
#include<stdio.h>
#include<stdlib.h>
#define size 10
int main() {
int a[3] = { 11, 22, 33 };
printf("\n a[0],value=%d : address=%u", a[0], &a[0]);
printf("\n a[1],value=%d : address=%u", a[1], &a[1]);
printf("\n a[2],value=%d : address=%u", a[2], &a[2]);
return (0);
}
```

Output:

```
a[0],value=11 : address=2358832
a[1],value=22 : address=2358836
a[2],value=33 : address=2358840
```

2. C Program for Reading and printing Array Elements

```
#include<stdio.h>
int main()
{
```

```
int i, arr[50], num;
printf("\nEnter no of elements :");
scanf("%d", &num);
//Reading values into Array
printf("\nEnter the values :");
for (i = 0; i < num; i++) {
scanf("%d", &arr[i]); }
//Printing of all elements of array
for (i = 0; i < num; i++) {
printf("\narr[%d] = %d", i, arr[i]); }
return (0);
}
```

Output:

```
Enter no of elements : 5
Enter the values : 10 20 30 40 50
arr[0] = 10
arr[1] = 20
arr[2] = 30
arr[3] = 40
arr[4] = 50
```

3. C Program to calculate addition of all elements in an array

```
#include<stdio.h>
int main() {
int i, arr[50], sum, num;
printf("\nEnter no of elements :");
scanf("%d", &num);
//Reading values into Array
printf("\nEnter the values :");
for (i = 0; i < num; i++)
scanf("%d", &arr[i]);
//Computation of total
```

```
sum = 0;
for (i = 0; i < num; i++)
sum = sum + arr[i];
//Printing of all elements of array
for (i = 0; i < num; i++)
printf("\na[%d]=%d", i, arr[i]);
//Printing of total
printf("\nSum=%d", sum);
return (0);
}
```

Output:

```
Enter no of elements : 3
Enter the values : 11 22 33
a[0]=11
a[1]=22
a[2]=33
Sum=66
```

4. C Program to find smallest element in an array

```
#include<stdio.h>
int main() {
int a[30], i, num, smallest;
printf("\nEnter no of elements :");
scanf("%d", &num);
//Read n elements in an array
for (i = 0; i < num; i++)
scanf("%d", &a[i]);
//Consider first element as smallest
smallest = a[0];
for (i = 0; i < num; i++) {
if (a[i] < smallest) {
smallest = a[i]; } }
```

```
// Print out the Result
printf("\nSmallest Element : %d", smallest);
return (0);
}
```

Output:

```
Enter no of elements : 5
11 44 22 55 99
Smallest Element : 11
```

5. C Program to find largest element in an array

```
#include<stdio.h>
int main() {
int a[30], i, num, largest;
printf("\nEnter no of elements :");
scanf("%d", &num);
//Read n elements in an array
for (i = 0; i < num; i++)
scanf("%d", &a[i]);
//Consider first element as largest
largest = a[0];
for (i = 0; i < num; i++) {
if (a[i] > largest) {
largest = a[i]; } }
// Print out the Result
printf("\nLargest Element : %d", largest);
return (0);
}
```

Output:

```
Enter no of elements : 5
11 55 33 77 22
Largest Element : 77
```

6. C Program to reverse an array elements in an array

```
#include<stdio.h>
int main() {
int arr[30], i, j, num, temp;
printf("\nEnter no of elements : ");
scanf("%d", &num);
//Read elements in an array
for (i = 0; i < num; i++) {
scanf("%d", &arr[i]);    }
j = i - 1;   // j will Point to last Element
i = 0;       // i will be pointing to first element
while (i < j) {
temp = arr[i];
arr[i] = arr[j];
arr[j] = temp;
i++;             // increment i
j--;          // decrement j
}
//Print out the Result of Insertion
printf("\nResult after reversal : ");
for (i = 0; i < num; i++) {
printf("%d \t", arr[i]); }
return (0);
}
```

Output:

```
Enter no of elements : 5
11 22 33 44 55
Result after reversal : 55 44 33 22 11
```

Multi Dimensional Array:

1. Array having more than one subscript variable is called Multi-Dimensional array.
2. Multi Dimensional Array is also called as **Matrix**.

Syntax: <data type> <array name> [row subscript][column subscript];

Example: Two Dimensional Arrays

Declaration: Char name[50][20];

Initialization:

```
int a[3][3] = { 1, 2, 3
                5, 6, 7
                8, 9, 0};
```

In the above example we are declaring 2D array which has 2 dimensions. First dimension will refer the row and 2nd dimension will refer the column.

Example: Three Dimensional Arrays

Declaration: Char name[80][20][40];

The following information are given by the compiler after the declaration

Example	Type	Array Name	Dimension No.	No. of Elements in Each Dimension
1	integer	roll	1	10
2	character	name	2	80 and 20
3	character	name	3	80 and 20 and 40

Two Dimensional Arrays:

1. Two Dimensional Array requires **Two Subscript Variables**
2. Two Dimensional Array stores the values in the form of matrix.
3. One Subscript Variable denotes the "**Row**" of a matrix.
4. Another Subscript Variable denotes the "**Column**" of a matrix.

	Column 0	Column 1	Column 2	Column 3
Row 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

Declaration and use of 2D Arrays:

```
int a[3][4];
```

```
for(i=0;i<row,i++)
for(j=0;j<col,j++) {
printf("%d",a[i][j]); }
```

Meaning of Two Dimensional Arrays:

1. Matrix is having 3 rows (i takes value from 0 to 2)
2. Matrix is having 4 Columns (j takes value from 0 to 3)
3. Above Matrix 3×4 matrix will have 12 blocks having 3 rows & 4 columns.
4. Name of 2-D array is '**a**' and each block is identified by the row & column number.
5. Row number and Column Number Starts from 0.

Two-Dimensional Arrays: Summary with Sample Example:

Summary Point	**Explanation**
No of Subscript Variables Required	2
Declaration	a[3][4]
No of Rows	3
No of Columns	4
No of Cells	12
No of for loops required to iterate	2

Memory Representation:

1. 2-D arrays are stored in contiguous memory location **row wise**.
2. 3 X 3 Array is shown below in the first Diagram.
3. Consider **3×3 Array is stored in Contiguous memory** location which starts from 4000.
4. Array element **a[0][0]** will be stored at address **4000** again **a[0][1]** will be stored to next memory location i.e. Elements stored row-wise
5. After **Elements of First Row are stored** in appropriate memory locations, elements of next row get their corresponding memory locations.

	Col 0	Col 1	Col 2
Row 0	1	2	3
Row 1	4	5	6
Row 2	7	8	9

6. This is integer array so each element requires 2 bytes of memory.

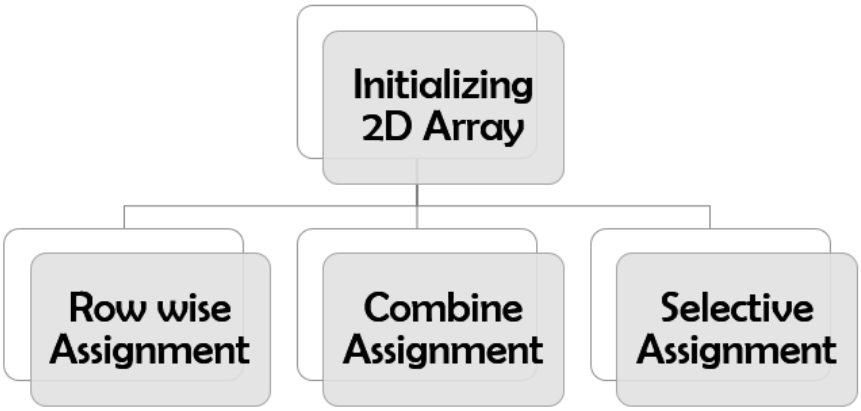
Basic Memory Address Calculation:

`a[0][1] = a[0][0] + Size of Data Type`

Element	Memory Location
a[0][0]	4000
a[0][1]	4002
a[0][2]	4004
a[1][0]	4006
a[1][1]	4008
a[1][2]	4010
a[2][0]	4012
a[2][1]	4014
a[2][2]	4016

1 4000	2 4002	3 4004
4 4006	5 4008	6 4010
7 4012	8 4014	9 4016

Initializing 2D Array



Method 1: Initializing all Elements row wise

For initializing 2D Array we need to assign values to each element of an array using the below syntax.

int a[3][2] = { {1, 4}, {5, 2}, {6, 5} };

Example Program

```
#include<stdio.h>
int main()
{
int i, j;
int a[3][2] = { { 1, 4 }, { 5, 2 }, { 6, 5 } };
for (i = 0; i < 3; i++) {
for (j = 0; j < 2; j++) {
printf("%d ", a[i][j]); }
printf("\n"); }
return 0;
}
```

Output:

```
1 4
5 2
6 5
```

We have declared an array of size 3 X 2, it contains overall 6 elements.

```
Row 1:  {1, 4},
Row 2:  {5, 2},
Row 3:  {6, 5}
We have initialized each row independently
a[0][0] = 1
a[0][1] = 4
```

Method 2: Combine and Initializing 2D Array

Initialize all Array elements but initialization is much straight forward. All values are assigned sequentially and row-wise

int a[3][2] = {1 , 4 , 5 , 2 , 6 , 5 };

Example Program:

```
#include <stdio.h>
int main() {
int i, j;
int a[3][2] = { 1, 4, 5, 2, 6, 5 };
for (i = 0; i < 3; i++) {
for (j = 0; j < 2; j++) {
printf("%d ", a[i][j]);  }
printf("\n"); }
return 0;
}
```

Output:

```
1 4
5 2
6 5
```

Method 3: Some Elements could be initialized

```
int a[3][2] = { { 1 }, { 5 , 2 }, { 6 } };
```

Now we have again going with the way 1 but we are removing some of the elements from the array. Uninitialized elements will get default 0 value. In this case we have declared and initialized 2-D array like this

```
#include <stdio.h>
```

```
int main() {
int i, j;
int a[3][2] = { { 1 }, { 5, 2 }, { 6 }};
for (i = 0; i < 3; i++) {
for (j = 0; j < 2; j++) {
printf("%d ", a[i][j]); }
printf("\n"); }
return 0;
}
Output:
1 0
5 2
6 0
```

Accessing 2D Array Elements:

1. To access every 2D array we requires **2 Subscript variables**.
2. i – Refers the **Row number**
3. j – Refers **Column Number**
4. a[1][0] refers element belonging to **first row and zeroth column**

Example Program: Accept & Print 2×2 Matrix from user

```
#include<stdio.h>
int main() {
int i, j, a[3][3];
// i : For Counting Rows
// j : For Counting Columns
for (i = 0; i < 3; i++) {
for (j = 0; j < 3; j++) {
printf("\nEnter the a[%d][%d] = ", i, j);
scanf("%d", &a[i][j]); } }
//Print array elements
for (i = 0; i < 3; i++) {
for (j = 0; j < 3; j++) {
```

```
printf("%d\t", a[i][j]); }
printf("\n"); }
return (0);
}
```

How it Works?

1. For Every value of row Subscript , the **column Subscript incremented from 0 to n-1 columns**
2. i.e. For Zeroth row it will accept zeroth, first, second column (a[0][0], a[0][1], a[0][2]) elements
3. In **Next Iteration** Row number will be incremented by 1 and the **column number again initialized to 0.**
4. **Accessing 2-D Array:** a[i][j] → Element From i^{th} Row and j^{th} Column

Example programs for practice:

1. C Program for addition of two matrices

2. C Program to find inverse of 3 X # Matrix

3. C Program to Multiply two 3 X 3 Matrices

4. C Program to check whether matrix is magic square or not?

1. C Program for addition of two matrices

```
#include<stdio.h>
int main() {
int i, j, mat1[10][10], mat2[10][10], mat3[10][10];
int row1, col1, row2, col2;
printf("\nEnter the number of Rows of Mat1 : ");
scanf("%d", &row1);
printf("\nEnter the number of Cols of Mat1 : ");
scanf("%d", &col1);
printf("\nEnter the number of Rows of Mat2 : ");
scanf("%d", &row2);
printf("\nEnter the number of Columns of Mat2 : ");
```

```
scanf("%d", &col2);
/* before accepting the Elements Check if no of rows and columns of
both matrices is equal */
if (row1 != row2 || col1 != col2) {
printf("\nOrder of two matrices is not same ");
exit(0); }
//Accept the Elements in Matrix 1
for (i = 0; i < row1; i++) {
for (j = 0; j < col1; j++) {
printf("Enter the Element a[%d][%d] : ", i, j);
scanf("%d", &mat1[i][j]); } }
//Accept the Elements in Matrix 2
for (i = 0; i < row2; i++)
for (j = 0; j < col2; j++) {
printf("Enter the Element b[%d][%d] : ", i, j);
scanf("%d", &mat2[i][j]); }
//Addition of two matrices
for (i = 0; i < row1; i++)
for (j = 0; j < col1; j++) {
mat3[i][j] = mat1[i][j] + mat2[i][j];}
//Print out the Resultant Matrix
printf("\nThe Addition of two Matrices is : \n");
for (i = 0; i < row1; i++) {
for (j = 0; j < col1; j++) {
printf("%d\t", mat3[i][j]); }
printf("\n"); }
return (0);
}
```

Output:

```
Enter the number of Rows of Mat1 : 3
Enter the number of Columns of Mat1 : 3
```

```
Enter the number of Rows of Mat2 : 3
Enter the number of Columns of Mat2 : 3
Enter the Element a[0][0] : 1
Enter the Element a[0][1] : 2
Enter the Element a[0][2] : 3
Enter the Element a[1][0] : 2
Enter the Element a[1][1] : 1
Enter the Element a[1][2] : 1
Enter the Element a[2][0] : 1
Enter the Element a[2][1] : 2
Enter the Element a[2][2] : 1
Enter the Element b[0][0] : 1
Enter the Element b[0][1] : 2
Enter the Element b[0][2] : 3
Enter the Element b[1][0] : 2
Enter the Element b[1][1] : 1
Enter the Element b[1][2] : 1
Enter the Element b[2][0] : 1
Enter the Element b[2][1] : 2
Enter the Element b[2][2] : 1
The Addition of two Matrices is :
2 4 6
4 2 2
2 4 2
```

2. C Program to find inverse of 3 X 3 Matrix

```
#include<stdio.h>
void reduction(float a[][6], int size, int pivot, int col) {
int i, j;
float factor;
factor = a[pivot][col];
for (i = 0; i < 2 * size; i++) {
```

```
a[pivot][i] /= factor; }
for (i = 0; i < size; i++) {
if (i != pivot) {
factor = a[i][col];
for (j = 0; j < 2 * size; j++) {
a[i][j] = a[i][j] - a[pivot][j] * factor; } } } }
void main() {
float matrix[3][6];
int i, j;
for (i = 0; i < 3; i++) {
for (j = 0; j < 6; j++) {
if (j == i + 3) {
matrix[i][j] = 1;}
else {
matrix[i][j] = 0; } } }
printf("\nEnter a 3 X 3 Matrix :");
for (i = 0; i < 3; i++) {
for (j = 0; j < 3; j++) {
scanf("%f", &matrix[i][j]); } }
for (i = 0; i < 3; i++) {
reduction(matrix, 3, i, i); }
printf("\nInverse Matrix");
for (i = 0; i < 3; i++) {
printf("\n");
for (j = 0; j < 3; j++) {
printf("%8.3f", matrix[i][j + 3]); } } }
```

Output:

```
Enter a 3 X 3 Matrix
1 3 1
1 1 2
2 3 4
```

```
Inverse Matrix
   2.000   9.000  -5.000
   0.000  -2.000   1.000
  -1.000  -3.000   2.000
```

3. C Program to Multiply two 3 X 3 Matrices

```
#include<stdio.h>
int main() {
int a[10][10], b[10][10], c[10][10], i, j, k;
int sum = 0;
printf("\nEnter First Matrix : ");
for (i = 0; i < 3; i++) {
for (j = 0; j < 3; j++) {
scanf("%d", &a[i][j]);  } }
printf("\nEnter Second Matrix :");
for (i = 0; i < 3; i++) {
for (j = 0; j < 3; j++) {
scanf("%d", &b[i][j]);  } }
printf("The First Matrix is : \n");
for (i = 0; i < 3; i++) {
for (j = 0; j < 3; j++) {
printf(" %d ", a[i][j]); }
printf("\n"); }
printf("The Second Matrix is : \n");
for (i = 0; i < 3; i++) {
for (j = 0; j < 3; j++) {
printf(" %d ", b[i][j]); }
printf("\n"); }
//Multiplication Logic
for (i = 0; i <= 2; i++) {
for (j = 0; j <= 2; j++) {
sum = 0;
```

```
for (k = 0; k <= 2; k++) {
sum = sum + a[i][k] * b[k][j]; }
c[i][j] = sum; } }
printf("\nMultiplication Of Two Matrices : \n");
for (i = 0; i < 3; i++) {
for (j = 0; j < 3; j++) {
printf(" %d ", c[i][j]); }
printf("\n"); }
return (0);
}
```

Output:

```
Enter First Matrix :
1    1    1
1    1    1
1    1    1
Enter Second Matrix :
2    2    2
2    2    2
2    2    2
The First Matrix is :
1    1    1
1    1    1
1    1    1
The Second Matrix is :
2    2    2
2    2    2
2    2    2
Multiplication Of Two Matrices :
6    6    6
6    6    6
6    6    6
```

Multiplication is possible if and only if

i. No. of Columns of Matrix 1 = No of Columns of Matrix 2

ii. Resultant Matrix will be of Dimension – c [No. of Rows of Mat1][No. of Columns of Mat2]

4. C Program to check whether matrix is magic square or not?

What is Magic Square?

1. A magic square is a simple mathematical **game developed during the 1500.**
2. Square is divided into **equal number of rows and columns.**
3. Start filling each square with the number from 1 to num (**where num = No of Rows X No of Columns**)
4. You can only **use a number once.**
5. Fill each square so that the **sum of each row is the same as the sum of each column.**
6. In the example shown here, **the sum of each row is 15,** and the sum of each column is also 15.
7. In this Example: The numbers from 1 through 9 is used only once. This is called a **magic square.**

```
#include<stdio.h>
#include<conio.h>
int main() {
int size = 3;
int matrix[3][3]; // = {{4,9,2},{3,5,7},{8,1,6}};
int row, column = 0;
int sum, sum1, sum2;
int flag = 0;
printf("\nEnter matrix : ");
for (row = 0; row < size; row++) {
for (column = 0; column < size; column++)
scanf("%d", &matrix[row][column]); }
printf("Entered matrix is : \n");
for (row = 0; row < size; row++) {
```

```
printf("\n");
for (column = 0; column < size; column++) {
printf("\t%d", matrix[row][column]); } }
//For diagonal elements
sum = 0;
for (row = 0; row < size; row++) {
for (column = 0; column < size; column++) {
if (row == column)
sum = sum + matrix[row][column]; } }
//For Rows
for (row = 0; row < size; row++) {
sum1 = 0;
for (column = 0; column < size; column++) {
sum1 = sum1 + matrix[row][column]; }
if (sum == sum1)
flag = 1;
else {
flag = 0;
break; } }
//For Columns
for (row = 0; row < size; row++) {
sum2 = 0;
for (column = 0; column < size; column++) {
sum2 = sum2 + matrix[column][row]; }
if (sum == sum2)
flag = 1;
else {
flag = 0;
break; } }
if (flag == 1)
printf("\nMagic square");
```

```
else
printf("\nNo Magic square");
return 0;
}
```

Output:

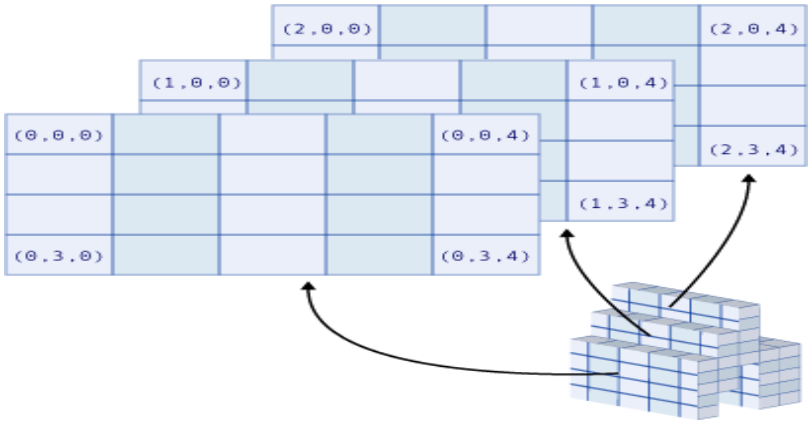
```
Enter matrix : 4 9 2 3 5 7 8 1 6
Entered matrix is :
 4 9 2
 3 5 7
 8 1 6
Magic square
Sum of Row1 = Sum of Row2 [Sum of All Rows must be same]
Sum of Col1 = Sum of Col2 [Sum of All Cols must be same]
Sum of Left Diagonal = Sum of Right Diagonal
```

Limitations of Arrays:

Array is very useful which stores multiple data under single name with same data type. Following are some listed limitations of Array in C Programming.

A. Static Data

1. Array is **Static data** Structure
2. Memory Allocated during **Compile time**.
3. Once Memory is allocated at Compile Time it cannot be changed during **Run-time**



B. Can hold data belonging to same Data types

1. Elements belonging to **different data types** cannot be stored in array because array data structure can hold data belonging to same data type.
2. **Example** : Character and Integer values can be stored inside separate array but cannot be stored in single array

C. Inserting data in an array is difficult

1. **Inserting element** is very difficult because before inserting element in an array we have to create empty space by shifting other elements one position ahead.
2. This operation is faster if the array size is smaller, but same operation will be more and more time consuming and non-efficient in case of array with large size.

D. Deletion Operation is difficult

1. Deletion is not easy because the elements are stored in contiguous memory location.
2. Like insertion operation , we have to delete element from the array and after deletion empty space will be created and thus we need to fill the space by moving elements up in the array.

E. Bound Checking

1. If we specify the size of array as 'N' then we can access elements up to 'N-1' but in C if we try to access elements after 'N-1' i.e. Nth element or N+1th element then we does not get any error message.
2. Process of checking the extreme limit of array is called Bound Checking and C does not perform **Bound Checking.**
3. If the array range exceeds then we will get garbage value as result.

F. Shortage of Memory

1. Array is Static data structure. Memory can be allocated at compile time only Thus if after executing program we need more space for storing additional information then we cannot allocate additional space at run time.
2. **Shortage of Memory** , if we don't know the size of memory in advance

G. Wastage of Memory

1. **Wastage of Memory**, if array of large size is defined

Applications of Arrays:

Array is used for different verities of applications. Array is used to store the data or values of same data type. Below are the some of the applications of array –

A. Stores Elements of Same Data Type

Array is used to store the number of elements belonging to same data type.

int arr[30];

Above array is used to store the integer numbers in an array.

arr[0] = 10;

arr[1] = 20;

arr[2] = 30;

arr[3] = 40;

arr[4] = 50;

Similarly if we declare the character array then it can hold only character. So in short character array can store character variables while floating array stores only floating numbers.

B. Array Used for maintaining multiple variable names using single name

Suppose we need to store 5 roll numbers of students then without declaration of array we need to declare following –

int roll1, roll2, roll3, roll4, roll5;

1. Now in order to get roll number of first student we need to access roll1.
2. Guess if we need to store roll numbers of 100 students then what will be the procedure.
3. Maintaining all the variables and remembering all these things is very difficult.

Consider the Array **int** roll[5]; Here we are using array which can store multiple values and we have to remember just single variable name.

C. Array can be used for Sorting Elements

We can store elements to be sorted in an array and then by using different sorting technique we can sort the elements.

Different Sorting Techniques are:

1. Bubble Sort
2. Insertion Sort
3. Selection Sort
4. Bucket Sort

D. Array can perform Matrix Operation

Matrix operations can be performed using the array. We can use 2-D array to store the matrix. Matrix can be multi dimensional.

E. Array can be used in CPU Scheduling

CPU Scheduling is generally managed by Queue. Queue can be managed and implemented using the array. Array may be allocated dynamically i.e at run time. [Animation will Explain more about Round Robin Scheduling Algorithm | Video Animation]

F. Array can be used in Recursive Function

When the function calls another function or the same function again then the current values are stores onto the stack and those values will be retrieving when control comes back. This is similar operation like stack.

Arrays as Function arguments:

Passing array to function:

Array can be passed to function by two ways:

1. **Pass Entire array**
2. **Pass Array element by element**

1. Pass Entire array

- Here entire array can be passed as a argument to function.
- Function gets **complete access** to the original array.
- While passing entire array address of first element is passed to function, any changes made inside function, directly **affects the Original value**.
- Function Passing method : **"Pass by Address"**

2. Pass Array element by element

- Here individual elements are passed to function as argument.
- Duplicate **carbon copy of Original variable** is passed to function.
- So any changes made inside function **do not affect the original value.**
- Function doesn't get complete access to the original array element.
- Function passing method is **"Pass by Value"**

Passing entire array to function:

- Parameter Passing Scheme : **Pass by Reference**
- Pass name of array as function parameter.
- Name contains the base address i.e. (Address of 0th element)
- Array values are updated in function.
- Values are reflected inside main function also.

Example Program #1:

```
#include<stdio.h>
#include<conio.h>
void fun(int arr[ ])
{
int i;
for(i=0;i< 5;i++)
 arr[i] = arr[i] + 10;
```

```
}
void main( )
{
int arr[5],i;
clrscr();
printf("\nEnter the array elements : ");
for(i=0;i< 5;i++)
scanf("%d",&arr[i]);
printf("\nPassing entire array .....");
fun(arr);  // Pass only name of array
for(i=0;i< 5;i++)
 printf("\nAfter Function call a[%d] : %d",i,arr[i]);
getch();
}
```

Output :

```
Enter the array elements : 1 2 3 4 5
Passing entire array .....
After Function call a[0] : 11
After Function call a[1] : 12
After Function call a[2] : 13
After Function call a[3] : 14
After Function call a[4] : 15
```

Passing Entire 1-D Array to Function in C Programming:

- Array is passed to function completely.
- Parameter Passing Method : **Pass by Reference**
- It is Also Called "**Pass by Address**"
- Original Copy is Passed to Function
- Function Body can modify **Original Value.**

Example Program #2:

```
#include<stdio.h>
#include<conio.h>
void modify(int b[3]);
void main()
{
int arr[3] = {1,2,3};
modify(arr);
```

```
for(i=0;i<3;i++)
printf("%d",arr[i]);
getch();
}
void modify(int a[3])
{
int i;
for(i=0;i<3;i++)
a[i] = a[i]*a[i];
}
```

Output:

```
1 4 9
```

Here “arr” is same as “a” because Base Address of Array “arr” is stored in Array “a”

Alternate Way of Writing Function Header:

void modify(int a[3]) **OR** void modify(int *a)

Passing Entire 2D Array to Function in C Programming:

Example Program #3:

```
#include<stdio.h>
void Function(int c[2][2]);
int main(){
int c[2][2],i,j;
printf("Enter 4 numbers:\n");
for(i=0;i<2;++i)
for(j=0;j<2;++j){
scanf("%d",&c[i][j]); }
Function(c);   /* passing multi-dimensional array to function */
return 0;
}
void Function(int c[2][2])
{
/* Instead to above line, void Function(int c[][2]) is also valid */
int i,j;
printf("Displaying:\n");
```

```
for(i=0;i<2;++i)
for(j=0;j<2;++j)
printf("%d\n",c[i][j]);
}
```

Output:

```
Enter 4 numbers:
2
3
4
5
Displaying:
2
3
4
5
```

Passing array element by element to function:

1. Individual element is passed to function using **Pass By Value** parameter passing scheme
2. An original Array element remains same as Actual Element is never passed to Function. Thus function body cannot modify **Original Value.**
3. Suppose we have declared an array 'arr[5]' then its individual elements are arr[0],arr[1]…arr[4]. Thus we need 5 function calls to pass complete array to a function.

Consider an array **int** arr[5] = {11, 22, 33, 44, 55};

Iteration	Element Passed to Function	Value of Element
1	arr[0]	11
2	arr[1]	22
3	arr[2]	33
4	arr[3]	44
5	arr[4]	55

Example Program #1:

```
#include< stdio.h>
#include< conio.h>
void fun(int num)
```

```
{
printf("\nElement : %d",num);
}
void main() {
int arr[5],i;
clrscr();
printf("\nEnter the array elements : ");
for(i=0;i< 5;i++)
scanf("%d",&arr[i]);
printf("\nPassing array element by element.....");
for(i=0;i< 5;i++)
fun(arr[i]);
getch();
}
```

Output:

```
Enter the array elements : 1 2 3 4 5
Passing array element by element.....
Element : 1
Element : 2
Element : 3
Element : 4
Element : 5
```

Disadvantage of this Scheme:

1. This type of scheme in which we are calling the function again and again but with **different array element is too much time consuming**. In this scheme we need to call function by pushing the current status into the system stack.
2. It is better to pass complete array to the function so that we can save some system time required for pushing and popping.
3. We can also pass the address of the individual array element to function so that function can modify the original copy of the parameter directly.

Example Program #2: Passing 1-D Array Element by Element to function

```
#include<stdio.h>
void show(int b);
void main() {
int arr[3] = {1,2,3};
int i;
for(i=0;i<3;i++)
show(arr[i]);
}
void show(int x)
{
printf("%d ",x);
}
```

Output:

```
1 2 3
```